基于 Spark 并行的密度峰值聚类算法 *

孙伟鹏[1]，吴锡生[1]，孟　斌[2]

(1. 江南大学 物联网工程学院，江苏 无锡 2141222; 2. 中船重工集团第七〇二研究所 软件工程中心，江苏 无锡 214082)

摘　要：针对 FSDP 聚类算法在计算数据对象的局部密度与最小距离时，由于需要遍历整个数据集而导致算法的整体时间复杂度较高的问题，提出了一种基于 Spark 的并行 FSDP 聚类算法 SFSDP。首先，算法通过空间网格划分将待聚类数据集划分成多个数据量相对均衡的数据分区；然后，利用改进的 FSDP 聚类算法并行地对各个分区内的数据执行聚类分析；最后，通过将分区间的局部簇集合并，生成全局簇集。实验结果表明，SFSDP 与 FSDP 算法相比能够有效地进行大规模数据集的聚类分析工作，并且算法在准确性和扩展性方面都有很好的表现。

关键词：聚类；密度峰值；空间划分；并行；Spark

中图分类号：TP301.6　　doi: 10.3969/j.issn.1001-3695.2018.04.0377

Spark-based parallel density clustering algorithm

Sun Weipeng[1], Wu Xisheng[1], Meng Bin[2]

(1. *School of IoT Engineering*, *Jiangnan University*, *Wuxi Jiangsu* 214122, *China*; 2. *Software Engineering Center*, *China Shipbuilding Group No*. 702 *Institute*, *Wuxi Jiangsu* 214082, *China*)

Abstract: In view of the problem that the overall time complexity of the FSDP clustering algorithm is high because the algorithm needs to traverse the entire data set when calculating the local density and minimum distance of data objects, this paper presents a Spark-based parallel FSDP clustering algorithm called SFSDP. First, the algorithm divides the dataset into multiple data partitions with relatively equal size by spatial meshing; then, the clustering analysis is performed on the data in each partition in parallel using the improved FSDP clustering algorithm; Global clusters are generated by grouping together local clusters between partitions. Experimental results show that SFSDP algorithm can effectively perform large-scale dataset clustering analysis compared with FSDP algorithm, and the algorithm has a good performance in terms of accuracy and scalability.

Key words: clustering; density peak; space division; parallel; Spark

0　引言

随着互联网在全球范围内的快速普及，人们每天都会面对来自社会、商业、医学、工程和科学以及人们日常生活各个方面的海量数据。数据的爆炸式增长、广泛可用和巨大规模把本文带入了一个真正的数据时代。而如何快速方便地从这些杂乱无章的大规模数据中挖掘出有用的信息，并将这些非结构化的数据转变成知识，这种需求催生了数据挖掘的诞生[1]。在数据挖掘领域中，聚类分析作为一种常用的数据分析方法，可以在对数据对象集的相关信息一无所知的情况下，将数据对象集合划分成多个簇，使得同一个簇中的对象彼此相似，但与其他簇中的对象不相似[2]。迄今为止，国内外众多研究学者针对各类应用场景已经提出了各式各样的聚类算法[3]，使得聚类分析的研究得到了很大的发展。在现实生活中，聚类分析早已经被广泛应用于多个领域中，包括网页搜索、决策分析、图像模式识别、文档摘要自动生成、自然语言处理、商务智能、生物学与安全等。

FSDP (clustering by fast search and find of density peaks)作为一种新的基于密度的聚类算法[4]，该算法将具有较大局部密度且互相距离较远的数据对象作为聚类中心，然后将每一个非中心数据对象沿着密度递增的最近邻方向迭代划分给相应的聚类中心，实现聚类划分。该算法被提出后得到了大量研究学者的关注与研究[5~8]。虽然 FSDP 算法具有算法原理简单、可以发现任意形状、大小的聚簇等优点，但该算法在面对数据量比较大或数据维度比较高的聚类任务时，由于 FSDP 算法在计算数据集中数据对象的局部密度和最小距离的时间复杂度较高，聚类的效率相对较低，所以算法不能很好地适用于大规模数据的聚类。

收稿日期：2018-04-24；**修回日期：**2018-06-11　　**基金项目：**国家自然科学基金资助项目（61672265）

作者简介：孙伟鹏（1990-），男，江苏连云港人，硕士研究生，主要研究方向为数据挖掘、云计算（sunwp244372610@163.com）；吴锡生（1959-），男，教授，博士，硕导，主要研究方向为人工智能、模式识别、云计算；孟斌（1980-），男，高级工程师，硕士研究生，主要研究方向为数据挖掘.

针对 FSDP 聚类算法在进行海量、高维度数据聚类时存在效率低、伸缩性差的问题，本文提出了一种基于 Spark 的并行 FSDP 聚类算法 SFSDP。该算法首先通过将海量数据集在空间中划分成多个数据量均衡的数据分区，然后将划分好的数据分区分发到集群中的计算节点上进行单独聚类，最后将局部聚类的结果合并生成全局聚簇集。实验结果表明，SFSDP 与 FSDP 算法相比，算法在保证准确率的前提下，具备较强的可扩展性，能够有效处理大规模数据集的聚类分析工作。

1　相关工作

1.1　FSDP 聚类算法

FSDP 算法认为聚类的中心应该同时具备以下两个特点：a）聚类中心相对于包围它的数据对象密度要大，即聚类中心被局部密度较低的邻居包围；b）聚类中心与任何具有更高局部密度对象之间的距离较远。对于数据集中每个数据点，需要计算数据点的局部密度 ρ_i 和距离具有更高局部密度数据点的最小距离 δ_i（下文简称为最小距离 δ_i）两个变量。其中局部密度 ρ_i 定义如式（1）所示。

$$\rho_i=\sum_j e^{-\left(d_{ij}/d_c\right)^2} \tag{1}$$

其中：d_{ij} 表示数据点 i 与 j 间的距离；d_c 表示截断距离。

最小距离 δ_i 表示的是数据点 i 与任何其他具有更高局部密度数据点之间的最小距离，其定义如式（2）所示。

$$\delta_i=\begin{cases}\min\limits_{j:\rho_j>\rho_i}\left(d_{ij}\right), i\ge 1\\ \max\limits_j\left(d_{ij}\right), \rho_i=\max(\rho)\end{cases} \tag{2}$$

通过计算数据集中的所有数据点的局部密度 ρ_i 和最小距离 δ_i，可得到数据集对应的 (ρ_i,δ_i) 分布。然后将该分布在平面坐标中画出，即可得到数据集对应的决策图。算法只需在图中选取同时具备较大 ρ 和 δ 的数据点作为聚类中心，然后将剩余的数据点归类到它邻近有更高局部密度的数据点所属的簇即可完成数据集聚类。

1.2　Spark 分布式框架

Spark[9,10]是由 UC Berkeley AMPLab 研发的一个快速而通用的并行计算框架。作为大数据时代最为主要的几个数据计算分析框架之一，Spark 为大数据分析应用提供了一个统一的数据处理平台，在这个平台下包含多个紧密集成的组件。其主要包含内核(spark core)部分和多个官方子模块（Spark SQL、Spark streaming、MLlib、GraphX）

Spark 具有高扩展和高容错等优点，并已经被广泛用于各种生成环境中。与 Hadoop 一样，Spark 也是为集群计算而设计的。但不同的是，Spark 可以把执行的中间结果缓存到内存中，这种方式可以大大提高算法的执行效率，因此，Spark 能够更好地适用于需要多次进行迭代的算法。并且与其他类似并行计算框架相比，Spark 不仅在运行速度和通用性能方面有很大的优势，而且在可扩展性和容错性方面也有较好的表现。

2　问题描述和相关定义

问题描述：

给定一组待聚类数据集，其所处的 d 维空间区域 DS 称为数据集 D 的数据空间（data space）。其中 p_i 表示 d 维空间中的数据对象，为数据对象 p_i 在第 k 维数轴上的投影。可通过 SFSDP 聚类算法得到一组聚簇集。

SFSDP 聚类算法的实现基于如下相关定义：

定义 1　空间网格、网格单元。将数据空间 DS 划分为多个互不重叠的区域，由这些区域构成的网格划分称为空间网格，记为 G。其中每个子区域为空间网格的一个网格单元，记为 g。

图 1 描述了基于二维数据空间 DS 的划分。其中，实线外边框围绕的大矩形代表数据空间 DS；实线内边框围绕的小矩形分别表示由 DS 划分的 g1、g2、g3、g4 网格单元。

定义 2　相邻网格单元。如果两个网格单元 g1 和 g2 所表示的空间区域在第 i 维是相邻区域，而在其他 d-1 维是同一区域，则 g1 和 g2 称为相邻网格单元。

定义 3　内部点。网格单元区域内的数据点称为内部点。

定义 4　临界点、临界区域。在网格单元区域内且与网格单元任意边界之间的距离小于密度截距 σ（见定义 6）的点称做临界点，临界点也属于内部点。涵盖临界点的区域称为临界区域。

定义 5　扩展点、扩展区域。在网格单元区域以外且与网格单元任意边界之间的距离小于密度截距 σ 的点称为扩展点。涵盖扩展点的区域称为扩展区域。

定义 6　改进的 ρ_i 计算公式、密度截距 σ、密度截距邻域)。由高斯函数的数学性质[11]可知，在计算数据对象的 ρ_i 时，对于给定的截断距离 d_c，当一个样本点距离指定数据点的距离大于 $3d_c/\sqrt{2}$ 时，则该样本点对指定数据点局部密度的计算影响非常小。这也就是说，每个数据点的局部密度主要取决于与其距离小于 $3d_c/\sqrt{2}$ 的近邻数据点。因此，在计算数据点局部密度时，可以只通过计算距离数据点 $3d_c/\sqrt{2}$ 范围内的邻近点来近似表示该数据点的局部密度。改进后数据点的局部密度计算公式如式（3）所示。

$$\rho_i=\sum_{d_{ij}\le\sigma} e^{-\left(d_{ij}/d_c\right)^2} \tag{3}$$

其中：$\sigma=3d_c/\sqrt{2}$，σ 称为密度截距。距离数据对象小于 σ 的范围称为数据对象的密度截距邻域。

数据空间网格划分如图 1 所示。由图 1 可知，对数据空间进行划分后，数据对象局部密度的计算只需要考虑数据对象所在网格单元内以及其相邻网格单元内的数据对象，有效避免了计算局部密度时需要遍历整个数据集的问题，大大降低了计算节点的工作负荷。

3　SFSDP 算法设计与实现

FSDP 算法在进行数据对象局部密度和最小距离计算时需

要遍历数据集中的所有数据对象，所以在进行大规模数据聚类时，如果只采用单节点处理，则节点的计算量会很大，导致算法的效率很低。本文提出的 SFSDP 聚类算法通过将大规模数据的聚类任务分解为若干可通过单节点处理的小规模任务来完成聚类，大大提高了数据集聚类的速度。SFSDP 算法的执行可分为三个阶段：

阶段 1 数据分区。根据数据集 D 在数据空间 DS 中的分布情况，通过空间划分将数据集 D 划分成若干规模较小且包含数据量大致均匀的数据分区。

阶段 2 局部聚类。各个计算节点在本地分区数据上执行改进的 FSDP 聚类算法，得到局部聚类结果。

阶段 3 局部簇合并。通过对局部聚类结果合并与调整，生成全局聚簇集。

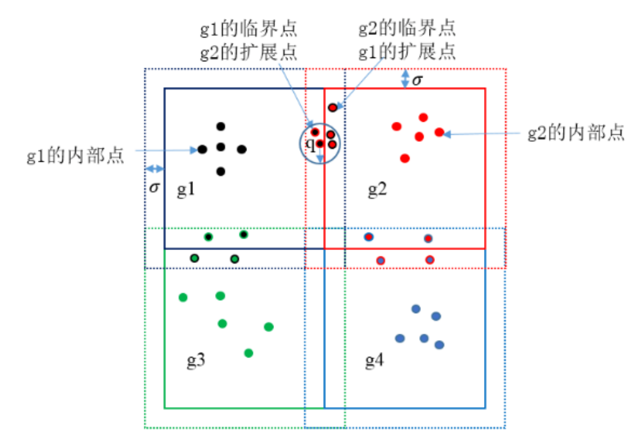


图 1 数据空间网格划分

3.1 数据分区

为了避免数据对象局部密度的计算需要遍历整个数据集，定义 6 给出了一种新的局部密度计算方法。改进后数据对象局部密度的计算只与数据空间中以该数据对象为中心、密度截距为半径范围内的数据对象有关，而与数据空间中其他的数据对象无关。因此，SFSDP 算法可通过将数据空间划分成空间网格，这样在计算数据对象局部密度时，只需考虑所处网格单元内和相邻网格单元内的数据对象，大大降低了算法的时间复杂度。在进行空间网格划分时，为了使得各个计算节点间负载均衡，防止数据倾斜现象的发生，SFSDP 算法采用基于网格单元内数据对象数目的划分策略，利用 KD-Tree[12]将数据空间划分成多个包含数据对象数目相对均衡的网格单元。

在对数据集分区时，对于划分完成后网格单元内的临界点，由于其密度截距邻域范围内的一部分邻近数据点与其不在同一个网格单元内，为了计算这些临界点的局部密度，在进行数据分区时，需要将一些数据对象同时分配给多个不同的分区。从图 1 可以看出，如果网格单元 g1 对应的分区 P 仅包含 g1 内的数据对象，由于临界点 q 的密度截距邻域内包含了网格单元 g2 中的数据对象，导致 q 的局部密度计算出错，所以对于与网格单元 g1 相对应的分区 P，需要将相邻网格单元 g2 内的一些数据对象也包含在分区 P 中。为了能更好地给出数据分区的完整定义，使用 p∈P 和 p<g 来分别表示数据对象 p 属于数据分区 P 和数据对象 p 包含在网格单元 g 中。

定义 7 网格单元的扩展σ空间。令 g 表示由数据空间 DS 通过空间网格划分得到的一个网格单元，将 g 各个维度的边界在空间中向外扩充密度截距σ大小的距离得到的空间区域称为网格单元 g 对应的扩展σ空间，记为 g+σ。

由定义 5 和 7 可以看出，网格单元的扩展σ空间即为网格单元包含的区域加上扩展区域。在图 1 中，由虚线围成的小矩形分别对应网格单元的扩充σ空间。

定义 8 数据分区。令 g+σ表示由数据空间 DS 通过空间网格划分得到的网格单元 g 的扩展σ空间，则 $P=\{p \mid p \in D \wedge p \prec g+\sigma\}$ 称为网格单元 g 对应的数据分区；相反，g 称为数据分区 P 对应的网格单元。

由定义 8 对数据分区的定义得知，一个数据分区包含了数据集中分布在对应网格单元扩展σ空间区域内的所有数据对象。此时，对于分区内的数据对象 p，如果 p 包含在分区对应的网格单元内，则可以通过定义 6 给出的公式计算出数据对象 p 的局部密度估计。

数据分区算法实现：

输入：D 为数据集；max 为网格单元内最大数据对象数目。

输出：Partitions 为数据集划分后得到的数据分区。

1. 通过数据集 D 得到数据空间 DS。
2. 将数据空间 DS 划分成若干个大小相等且互不重叠的网格单元。
3. 将数据对象映射到网格单元，并计算每个网格单元内数据对象的个数。
4. 初始化一个待分割空间队列 Queue，并将 DS 添加到队列中。
5. 初始化一个空的网格单元集合 G。
6. 弹出待分割空间队列中的队首空间区域 S。
7. 计算数据空间区域 S 内包含的数据对象数目 n。如果 n 小于 max，则将 S 添加到集合 G 中；否则，求得空间区域 S 内的数据对象在 d 维空间中各个维度的方差，选取方差最大的维度作为分割维度，将 S 划分成两个包含数据量均衡的子空间区域 S1、S2，并将 S1 和 S2 添加到待分割空间队列中
8. 如果队列 Queue 为空，则将 G 作为数据集 D 的空间网格划分；否则跳转到步骤 6。
9. 根据划分好的空间网格 G，由定义 8 得到数据集的数据分区

3.2 分区内局部聚类

在数据集上完成数据分区工作后，为了使各个计算节点可以并行在对应的数据分区上执行局部聚类，需要对原始 FSDP 算法进行两处修改：

a）FSDP 在计算数据点局部密度和最小距离时需要遍历整个数据集，为了可以在分区内独立计算这两个结果，这里分别采用公\式（3）和（4）在分区 P 内计算数据对象的局部密度和最小距离。

$$\delta_i=\begin{cases}\min\limits_{j:\rho_j>\rho_i}\left(d_{ij}\right), i\geq 1\\ \max\limits_{j}\left(d_{ij}\right), \rho_i=\max(\rho)\end{cases} p_i, p_j\in P \tag{4}$$

b）FSDP 算法需要人为参与聚类中心的选取，为了摆脱算法的人为干预，本文采用式（5）作为辅助函数。算法通过为局部聚类设定一个聚类中心阈值，然后将分区内各个数据对象的γ取值与设定的阈值作比较，将γ取值大于设定阈值的数据对象作为聚类中心的候选对象。

$$\gamma_i=\rho_i*\delta_i,(i=1,2,\cdots,n) \tag{5}$$

局部聚类算法实现：

输入：P 为数据分区；d_c 为截断距离；threshold 为聚类中心阈值。

输出：数据分区的局部聚类结果。

1. for each point in P do
2. 　利用式（3）计算数据点的局部密度估计值 ρ_i
3. end for
4. for each point in P do
5. 　利用式（4）计算数据点的最小距离和最近邻点
6. end for
7. for each point in P do
8. 　利用式（5）计算分区内数据对象的 γ_i 取值
9. end for
10. for each point in P do
11. 　比较数据点 γ_i 的取值与 threshold 的大小关系，如果 γ_i>threshold，则该数据点选为聚类中心，并给定所属簇
12. for each point in P do
13. 　if point not centerpoint then
14. 　　将数据点划分给最近高密度邻点所属的簇
15. 　end if
16. end for
17. 计算每个聚簇的边界密度
18. for each point in P
19. 　如果数据对象的局部密度大于所属簇边界密度，则标记该数据对象为核心点，否则为光晕点
20. 输出聚类结果

3.3　局部簇合并

为了使得 SFSDP 聚类算法在局部聚类阶段各个计算节点可以独立在对应的数据分区上进行聚类，数据分区阶段将数据集划分成了若干组互相重叠的数据分区。这些相互重叠的分区内包含了一些公共的数据对象。在局部聚类合并阶段，算法可以通过评估这些公共数据对象（即临界点和扩展点）的特征，找出所有需要合并的局部簇。

为了描述的方便，假设 C1 和 C2 分别表示两个重叠分区 P1 和 P2 通过局部聚类得到的一个局部簇(即 C1⊆P1，C2⊆P2，且 P1∩P2≠Φ。)

定理 1　合并点定理。如果存在数据点 p∈C1∩C2，且 p 在局部簇 C1 和 C2 中都是核心成员，则需要合并 C1 和 C2（即局部簇 C1 中的点和局部簇 C2 中的点应该归属于同一个全局簇）。数据点 p 称为局部簇 C1 和 C2 的合并点。

证明　核心成员对应簇的中心，由高密度的数据对象组成。上述定理通过聚簇核心成员的定义可以很容易得出。

定理 2　所有的合并点一定是分区内的临界点或扩展点。如果数据点 p 是局部簇 C1 和 C2 的一个合并点，那么数据点 p 一定是分区内的临界点或扩展点。

证明　因为由合并点 p∈C1∩C2 可以得出 p∈P1∩P2，所以点 p 位于 P1 和 P2 的重叠区域。由临界点和扩展点的定义，定理 2 得证。

基于以上定理，在确定局部聚类的合并点时，只需要获取分区内所有的临界点和扩展点，并判断其是否满足定理 1 即可。为了获取到分区内所有的临界点和扩展点，由图 1 可以看出，一个分区的临界点同时也是相邻分区的扩展点，且只有扩展点和临界点才会出现在多个分区内，所以可以通过对局部聚类结果按照数据对象进行分组，然后通过过滤组内成员个数大于 1 的数据点作为合并点的候选对象，最后通过观察候选对象是否满足定理 1 来确定所有的合并点。

在找到所有的合并点后，根据合并点在不同分区中所属的局部簇可以得到不同分区间局部簇的关联关系。为了可以方便地给局部簇分配全局簇标识，利用图论知识将局部簇与其之间的关联关系分别表示成图的顶点和边，最终构建出一张局部簇关联图，属于同一个全局簇的局部簇集合对应了图中的一个连通子图。通过为图中各个连通子图的顶点所代表的局部簇生成一个全局聚簇标识即可完成局部簇标志到全局簇标志的映射。

局部簇合并算法实现：

输入：D 为局部聚类结果，数据格式为 Key-Value：(p，(pId，clusterId，flag))。其中 p 表示数据对象本身；pId 表示数据分区 Id；clusterId 表示分区内局部簇 Id；flag 表示数据对象是核心成员还是光晕成员。

输出：全局聚类结果。

1. 对数据集 D 按照数据对象 p 进行分组，通过过滤组内成员大于 1 的数据对象得到合并点候选对象集合 CS。
2. 遍历合并点候选对象集合 CS，通过观察候选对象在各分区局部簇中的标记（flag，是否为核心成员）和定理 1 判断候选对象是否为合并点，得到合并点集合 MS。
3. 根据合并点所属的局部簇类，构建局部簇关联关系集合 LS。集合元素格式为(p1.clusterId，p2.clusterId)，代表 p 同时属于分区 p1 中标志为 clusterId 的局部簇和分区 p2 中标志为 clusterId 的局部簇。
4. 初始化局部簇关联图 G，将所有局部簇作为顶点添加到图 G 中，同时为每个顶点生成一条指向自身的边。
5. 遍历集合 LS，对于集合中的每个元素(p1.clusterId，p2.clusterId)，如果在关联图 G 中不存在相对应的边，那么

将(p1.clusterId, p2.clusterId)作为图 G 的一条边加入到图中。最终，所有属于同一全局簇的局部簇构成了图 G 的一个连通子图。

6. 为图 G 中的每个连通子图 g 生成一个全局簇标志，将 g 中所有顶点代表的局部簇标志都映射到这个全局聚簇标志，从而建立局部簇标识到全局簇标识的映射。
7. 利用得到的局部簇标志到全局簇标志的映射更新每个局部簇中数据对象的簇标志。

3.4 算法时间和空间复杂度分析

1）时间复杂度分析

假设待聚类数据集包含 n 个数据对象，FSDP 聚类算法的时间消耗主要包含两个阶段：a）计算数据对象的局部密度阶段；b）计算数据对象的最小距离阶段。由于 FSDP 算法在这两个阶段的计算都需要两层循环遍历数据集，所以算法的整体时间复杂度为 $O(n^2)$。而 SFSDP 聚类算法的时间消耗主要包含三个阶段：a）数据分区阶段；b）分区内局部聚类阶段；c）局部簇合并阶段。假设集群中一共有 M 个计算节点并行处理，在数据分区阶段，需要对数据集进行多次遍历来确定数据划分，时间复杂度为 $O(n/m)$。在局部聚类阶段，假设指定每个网格单元内数据对象的数量最多不超过 m，则数据集在数据分区阶段可大致划分成 n/m 个数据分区，每个分区内数据对象的数量大致为 m（因为分区内包含了网格单元的扩展点，所以实际分区内的数据对象数量可能会大于 m。这里由于网格单元的扩展点数量相对于 m 过小，所以可以忽略）。在每个分区上执行聚类的时间复杂度为 $O(m^2)$，所以该阶段的时间复杂度为 $O((m*n)/M)$。在局部簇合并阶段，需要遍历数据集来查找局部簇的合并点和更新数据对象的全局簇标志，时间复杂度为 $O(n/M)$。所以 SFSDP 算法最后总的时间复杂度计算公式如式（6）所示。考虑到计算节点间的通信和数据传输需要额外的代价，算法的实际时间复杂度会略大于这里给出的计算。

$$T = O(n/M) + O((n*m)/M) + O(n/M) \tag{6}$$

2）空间复杂度分析

假设待聚类数据集包含 n 个数据对象，由于需要记录数据集中每一对数据对象之间的距离，所以 FSDP 聚类算法的空间复杂度为 $O(n^2)$。而在 SFSDP 中，假设指定每个网格单元内数据对象的数量最多不超过 m，聚类算法将数据集划分到 n/m 个分区中（这里是理想状态，实际分区的数量会大于等于 n/m）。因此，SFSDP 聚类算法对每个计算节点的存储空间要求为 $O(m^2)$。

4 仿真实验

4.1 实验环境搭建

本实验通过配置五台普通机器搭建 Spark 集群，其中每台计算机的硬件配置为 Intel Core i5 2.7 GHz 4 核 CPU，8 GB 内存，512 GB 硬盘。软件环境为 CentOS 6.5 操作系统。所有的计算节点通过千兆以太网交换机互连。Spark 版本为 2.3.0。

4.2 算法准确性对比实验

为了验证 SFSDP 算法的准确性，实验选取表 1 中给出的测试数据集分别对 SFSDP 和 FSDP 算法进行聚类，得到的各个测试数据集的准确性对比实验结果如表 2 所示。

表 1　SFSDP 与 FSDP 算法聚类准确性

数据集	样本数目	属性维度	类别个数
Spiral	312	2	3
R15	600	2	15
Aggregation	788	2	7
D31	3100	2	31
Iris	150	4	3
Wine	178	13	3
Glass	214	9	6
Seeds	210	7	3
WDBC	596	30	2

表 2　FSDP 与 SFSDP 算法聚类准确性

数据集	FSDP	SFSDP
Spiral	1	1
R15	99.66%	99.24%
Aggregation	99.47%	94.52%
D31	96.16%	92.31%
Iris	87%	83.47%
Wine	70.78%	65.39%
Glass	57.34%	55.76%
Seeds	88.57%	86.35%
WDBC	57.47%	53.65%

由表 2 可以看出，SFSDP 算法的聚类准确性相比于原始 FSDP 聚类算法略低。经过分析得知其主要原因有两个：a)在计算分区内扩展点的局部密度时，由于其密度截距范围内有一部分邻近对象不在同一个分区中，导致扩展点的局部密度计算偏小，可能致使扩展点被误判成光晕成员；b)由于在分区内判断数据对象的最小距离，对于有些边界点，其全局高密度最邻近数据点可能出现在相邻分区中，导致数据点划分出错。

4.3 算法性能对比实验

为了度量 SFSDP 算法与 FSDP 算法之间的性能差距，实验从香港科技大学智能城市研究小组（http://www.cse.ust.hk/scrg/）获取实验测试数据集。该测试数据集信息包含了 4 000 多辆出租车的行驶数据。通过对原始文件处理，得到只含有车辆坐标的数据文件。从生成的坐标文件中随机选取生成大小不一的五个数据样本 D1（1 千个坐标点）、D2（2 千个坐标点）、D3（5 千个坐标点）、D4（1 万个坐标点）、D5（2 万个坐标点）。SFSDP 算法和 FSDP 算法在上述五个测试集上执行的性能对比实验结果如图 2 所示。

从图 2 可以看出，FSDP 算法由于无须对数据集进行分区和进行计算节点间的通信，在数据规模较小时算法用时较少；但当数据规模快速增长时，FSDP 算法运行的时间会快速增长，

在数据规模达到 20 000 时，FSDP 算法就已经不能有效地完成聚类分析，因此算法的伸缩性较差。而对于 SFSDP 算法来说，由于 SFSDP 算法需要在数据集分区和节点间通信上花费一定的时间，当数据规模较小时，数据集分区和节点间通信所占的时间比重较高，算法的效率劣于 FSDP 算法；但当数据规模上升时，随着数据分区及节点间通信时间比重的下降，SFSDP 算法的整体花费时间增长平缓，因此算法具有较强伸缩性。

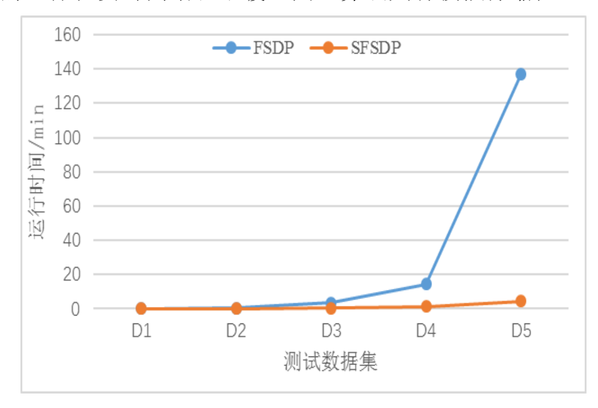


图 2　SFSDP 和 FSDP 算法运行的时间对比

4.4　加速比和扩展比分析

为了更好地验证 SFSDP 算法的可扩展性，实验从车辆坐标数据文件中随机选取生成五个测试数据集：D1（1 万个坐标点）、D2（10 万个坐标点）、D3（100 万个坐标点）、D4（200 万个坐标点）和 D5（500 万个坐标点）数据集。通过在不同数量节点下对各个测试集进行实验，计算对应测试数据集的加速比和扩展比。实验结果分别如图 3 和 4 所示。

从图 3 可以看出，SFSDP 算法在进行聚类处理时，随着节点数量的增加，加速比也随之增大。但对于小规模数据集 D1 和 D2，随着节点的增加，加速比曲线斜率下降比较明显。这主要是因为 D1 和 D2 的数据集规模较小，随着节点数目的增加，集群用于计算的时间比较小，而用于节点间通信的时间比较大。而对于大规模数据集 D3、D4、D5，加速比更接近线性增加。在图 4 中，随着集群节点个数的增加，算法在各个数据集上的扩展比逐渐减小。这主要是因为随着集群中节点个数的增加，使得节点之间的通信代价增加。当数据规模越大时，SFSDP 算法的扩展比越高，这表明算法在大规模数据上可以取得较好的扩展性。

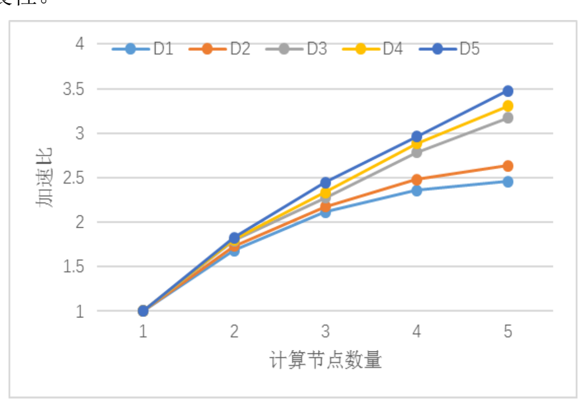


图 3　SFSDP 算法在不同规模测试集上的加速比

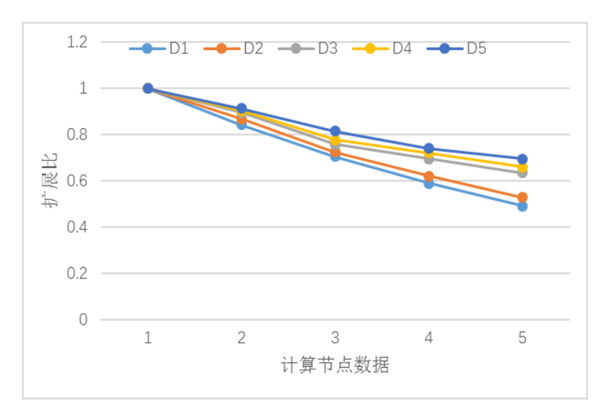


图 4　SFSDP 算法在不同规模测试集上的扩展比

5　结束语

针对 FSDP 聚类算法在进行海量、高维度数据聚类时存在效率低、伸缩性差的问题，为了适应大规模数据聚类分析，本文提出了一种基于 Spark 的并行 FSDP 聚类算法 SFSDP。算法首先通过将海量数据集在空间中划分成多个数据量均衡的数据分区；然后利用多个计算节点在各个划分好的数据分区上并行执行改进后的 FSDP 聚类算法，得到各个数据分区的局部聚类结果；最后将局部聚类的结果合并生成全局聚类簇集。实验结果表明，SFSDP 算法与 FSDP 算法相比，算法在保证准确率的前提下，具备较强的可扩展性，能够有效处理大规模数据集的聚类

参考文献：

[1] Han Jiawei, Kamber M. 数据挖掘概念与技术 [M]. 范明，译. 北京：机械工业出版社, 2001: 232-236.

[2] Zhou Z H. Three perspectives of data mining [J]. Artificial Intelligence, 2003, 143 (1): 139-146.

[3] Omran M G H, Engelbrecht A P, Salman A. An overview of clustering methods [J]. Intelligent Data Analysis, 2007, 11 (6): 583-605.

[4] Rodriguez A, Laio A. Clustering by fast search and find of density peaks [J]. Science, 2014, 344 (6191): 1492-1496.

[5] 淦文燕，刘冲. 一种改进的搜索密度峰值的聚类算法 [J]. 智能系统学报，2017, 12 (2): 229-236. (Gan Wenyan, Liu Chong. An improved clustering algorithm that searches and finds density peaks [J]. CAAI Transactions on Intelligent Systems, 2017, 12 (2): 229-236.)

[6] 蒋礼青，张明新. 郑金龙. 快速搜索与发现密度峰值聚类算法的优化研究 [J]. 计算机应用研究, 2016, 12 (11): 3251-3254. (Jiang Liqing, Zhang Mingxin, Zheng Jinlong, *et al.* Optimization of clustering by fast search and find of density peaks [J]. Application Research of Computers, 2016, 12 (11): 3251-3254.)

[7] 贺玲，吴玲达，蔡益朝. 数据挖掘中的聚类算法综述 [J]. 计算机应用研究, 2007, 24 (1): 10-13 (He Ling, Wu Lingda, Cai Yichao. Survey of clustering algorithms in data mining [J]. Application Research of Computers, 2007, 24 (1): 10-13.)

[8] Bie R, Mehmood R, Ruan S, *et al.* Adaptive fuzzy clustering by fast search

and find of density peaks [J]. Personal & Ubiquitous Computing, 2016, 20 (5): 785-793.

[9] Zaharia M, Chowdhury M, Franklin M J, *et al.* Spark: cluster computing with working sets [C]// Proc of Usenix Conference on Hot Topics in Cloud Computing. [S. l.] : USENIX Association, 2010: 10-10.

[10] Zaharia M, Chowdhury M, Das T, *et al.* Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing [C]// Proc of Usenix Conference on Networked Systems Design and Implementation. [S. l.] : USENIX Association, 2012: 2-2.

[11] Barany I, Vu V H. Central limit theorems for Gaussian polytopes [J]. Annals of Probability, 2006, 35 (4): 1593-1621.

[12] Bentley J L. Multidimensional binary search trees used for associative searching [J]. Communications of the ACM, 1975, 18 (9): 509-517.